7. Lorem Ipsum

Program Name: Lorem.java Input File: lorem.dat

In order to graduate, Joel has to pass the history class he is in. To pass his history class, Joel has to submit 9 different papers during the semester. Joel spent hours working on each of the papers, but realized that he always seemed to get the same grade no matter how much effort he put in. It dawned on him that none of the TA's actually read any of the student's papers and instead arbitrarily assigned grades. In order to test this hypothesis, Joel wants to submit a fake paper. He knows that just printing out a bunch of random filler text would stand out too much, so Joel wants to generate text that looks closer to English. He has thought about using a Markov chain text generator.

The basic premise is to first analyze a piece of known text and determine the probabilities of each word appearing after the previous *n* words and then generate the random text by choosing a random seed and selecting the words one at a time based on the previous *n* generated words.

For example, if the word *sunset* occurs thrice (three times) as often as the word *man* after the word *beautiful* (and no other words appear following *beautiful*), then if we use n=1, the probability of *beautiful sunset* is 0.75, while the probability of *beautiful man* is only 0.25.

Input

The first line of input contains an integer C that is the number of cases to follow. The first line of input in each case contains T, the number of sentences in the source text. The next T lines each represent a single sentence of the known text Joel wants to analyze and use for likeness. A sentence is a series of space delimited lowercase words. Assume the sentences are not connected, so the last word of a sentence does not lead to the first word of the next sentence.

Output

For each test case, print the probabilities of all possible word pairs. That is, for each unique word, print every word that can follow it and the respective probabilities, rounded to the nearest hundredth. If no words follow it, then omit the word. The lines should be sorted lexicographically by the unique word on that line. The listing for each word should be sorted by probability of occurrence and then also lexicographically if two words have equal probability.

Constraints

```
1 <= T <= 50
1 <= Number of unique words <= 100
```

Example Input File

```
3
3
a beautiful sunset that was mistaken for a dawn
that was a beautiful sunset
a beautiful man
4
if ever i would leave you
i would never leave you today
if i leave will you leave too
if i will leave you leave too
2
she loves me she loves me not she loves me she loves me not
she loves him more than she loves me or she loves you
```

Example Output to Screen

```
a: 0.75,beautiful 0.25,dawn
beautiful: 0.67,sunset 0.33,man
for: 1.00,a
mistaken: 1.00,for
sunset: 1.00,that
that: 1.00,was
was: 0.50,a 0.50,mistaken

ever: 1.00,i
i: 0.50,would 0.25,leave 0.25,will
if: 0.67,i 0.33,ever
leave: 0.50,you 0.33,too 0.17,will
never: 1.00,leave
will: 0.50,leave 0.50,you
would: 0.50,leave 0.50,never
you: 0.67,leave 0.33,today

him: 1.00,more
loves: 0.71,me 0.14,him 0.14,you
me: 0.40,not 0.40,she 0.20,or
more: 1.00,than
not: 1.00,she
or: 1.00,she
she: 1.00,loves
than: 1.00,she
```

Explanation of Output

In the first example the word *a* was thrice followed by *beautiful* and once followed by *dawn*, so the probability of *beautiful* following *a* is 0.75, while *dawn* following *a* is only 0.25. The words *dawn* and *man* are never followed by another word so they are omitted in the example output

In the second example on the second line *leave* and *will* both have a probability of .25 so they are printed lexicographically left to right.